Data Platform EAC PoCs design proposal

The following document describes the proposed design for the Data Platform environment for the EAC Proof of Concepts (PoC).

Table of contents

Introduction

Stakeholders

The following stakeholders are identified for the pwc-eac-pilots project:

Stakeholder	Description	Contacts
PwC team	Responsible for developing the PoC	christophe.cop@pwc.com alexis.laks@pwc.com
DIGIT D.1 Data Platform technical team	Responsible for setting up the infrastructure on which the PoC will be developed and deployed	bradeschouwer@deloitte.com bverhoeve@deloitte.com
DIGIT D.1 Data Platform coordination	Responsible for project management	Emmanouil.MARAGKOUDAKIS@ext.ec.europa.eu

Introduction

Two pwc-eac-pilots data science projects have been proposed, which will focus on the relationship between the current market demand for certain skills to university courses that focus on developing these skills with their students. The goal is to on one hand gather top journal papers of several industry sectors that define the need or use of certain skill sets or profiles (e.g. management, economics, law, computer science, etc.). From these papers, the most in demand skills will be extracted. On the other hand, data will be gathered from the course sites of various universities, from which the skills towards they contribute, will be extracted.

A word-to-vector model will be built from both datasets, which will than be matched to investigate the current correlation between both the market demand for skills and the offering of universities.

PoC 1

The first PoC focusses on the data gathered from universities. This data contains university course descriptions, which will be matched to relevant skills. The cosine similarity between the course description and all skills is calculated, effectively indicating which course matches with which skills.

The data from several universities has been scraped. On the infrastructure, the data will first be pre-processed by dropping NaNs, removing duplicates and constructing a vector representation of the strings (both course descriptions and skills). The vector representation of the strings is built using the pre-trained Google word2vec model (approximate size: 1.5 GB raw). From the word vectors, the cosine similarity between each course description and skill is calculated, and saved in a matrix. The calculations have already been implemented in Python using Pandas.

This matrix can then be used to construct a graph, which links courses and relevant skills (nodes) with each other with the cosine similarity as value on the edges. The graph is then visualized using R Shiny, which is published as a web application. With the web application, users can then explore the results of the PoC.

Two phases are identified for the PoC:

1. **Training**: Data has already been gathered and preprocessed. The training phase of the project will focus on preparing the word-vector matrix and calculating the cosine similarity.
2. **Deployment**: Based on the model from the first phase of the project, the model will be deployed and used in production to showcase the project. A user interface which integrates with the model will be built, through which the users can explore and interact with the data and the results.

Requirements

The following section describes the initial requirements provided by the stakeholders.

Data sources

The dataset used contains the course descriptions from universities and relevant skills. The dataset has the following attributes:

Parameter	Value
Data type	Structured
Size	approximately 500 MB
Data format	csv
Streaming	No
Compressed	No
Location	Unknown
Confidentiality	Public

Computational requirements

The current implementation has already been deployed on a personal workstation, which provided a single core CPU. The application code is currently written for a single thread and doesn't take advantage of multithreading across multiple CPU cores. This can be optimized further by the application code.

Therefore, higher numbers of CPU cores are not stringent requirements, since no tangible argument is present that a high amount of cores will improve performance. A general purpose amount of vCPUs will be used.

Memory requirements

Several data structures will have to be kept in memory for the PoC. The following components while require memory in the training phase:

Component	Absolute size (GB)	Description
University data	0.5	Scraped course description data
ESCO data	0.5	Skills description data
word2vec model	6	Pre-trained Google word2vec model. The model is larger in memory than when stored on disk due to additional memory being allocated by the Python libraries.
word vector of data	0.5	Resulting word vector of the data
Cosine similarity matrix	1	Cosine similarity matrix of vector data
OS and apps	6	Contingency breathing memory for OS and apps
Total	14.5	Total minimal memory required

This total memory will be rounded up to 16 GB, due to the memory of AWS instances increasing in increments of factor 2.

During the deployment phase, data structures as the word2vec model no longer require memory, resulting in a smaller memory requirements:

Component	Absolute size (GB)	Description
Graph data	1	Graph representation of course, skill data and their cosine similarities
OS and apps	6	Contingency breathing memory for OS and apps
Total	7	Total minimal memory required

This total memory will be rounded up to 8 GB, due to the memory of AWS instances increasing in increments of factor 2.

Storage requirements

The stakeholders have indicated to have at least a disk size of 200 GiB.

Scaling requirements

The stakeholders have expressed that although the project is currently in a proof of concept phase with a small amount of data; it is foreseen to scale up the project to larger amounts of data in future iterations.

Therefore, the stakeholders have expressed the needs to move away from an implementation which is specified to a single machine and use frameworks which can also run on a computing cluster.

Required software

During the training phase the following software packages are required:

Software	Version	Description
Python	3.7.3	Python is an interpreted, high-level, general-purpose programming language.
Jupyter Notebook	4.4.0	The Jupyter Notebook is an open-source web application that allows you to create and share documents that contain live code, equations, visualizations and narrative text.
Conda	4.6.14	Package manager, comes with the Anaconda stack, including popular Python data science packages such as Pandas, scikitlearn etc.
Apache Spark	2.4.3	Distributed processing framework and programming model that helps you do machine learning, stream processing, or graph analytics.
Hadoop	2.7	Software framework for distributed computing

During the deployment phase the following software packages are required:

Software	Version	Description
R	3.6.0	Programming language and free software environment for statistical computing and graphics supported by the R Foundation for Statistical Computing.
Shiny Server	1.5.9.923	Open source R package that provides an elegant and powerful web framework for building web applications using R.

Security

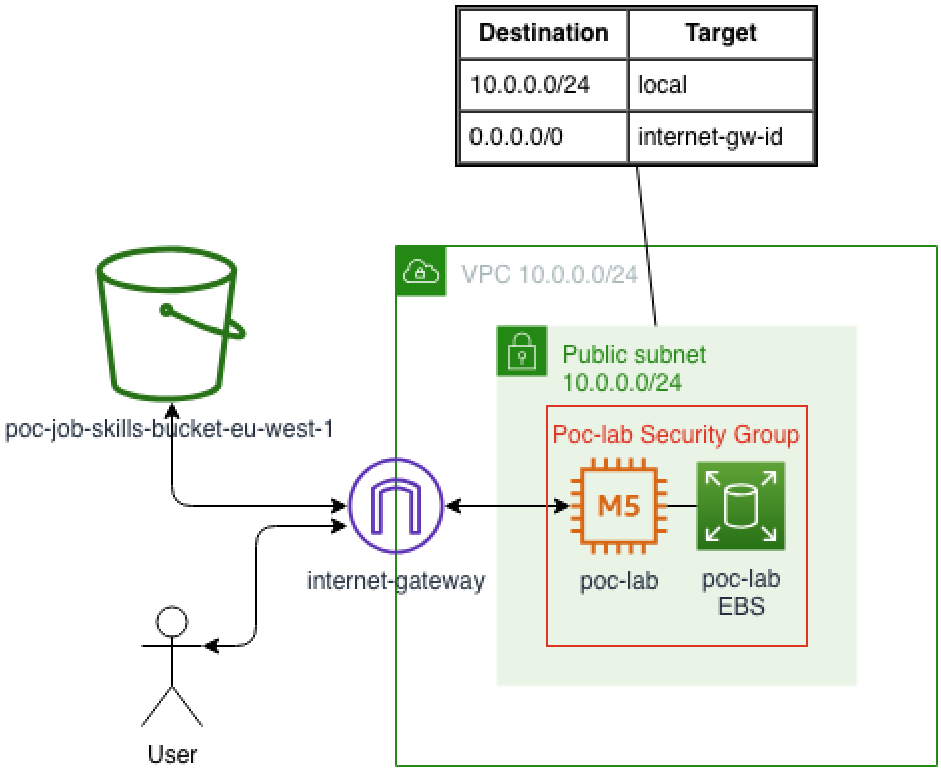
The data and code is marked as public by the stakeholders, strict security requirements are not required.

The environment may be publicly reachable. Encryption at rest/in transit is not required.

A lightweight authentication mechanism using the existing software tools (Shiny, Jupyter) will be provided.

Proposed solution

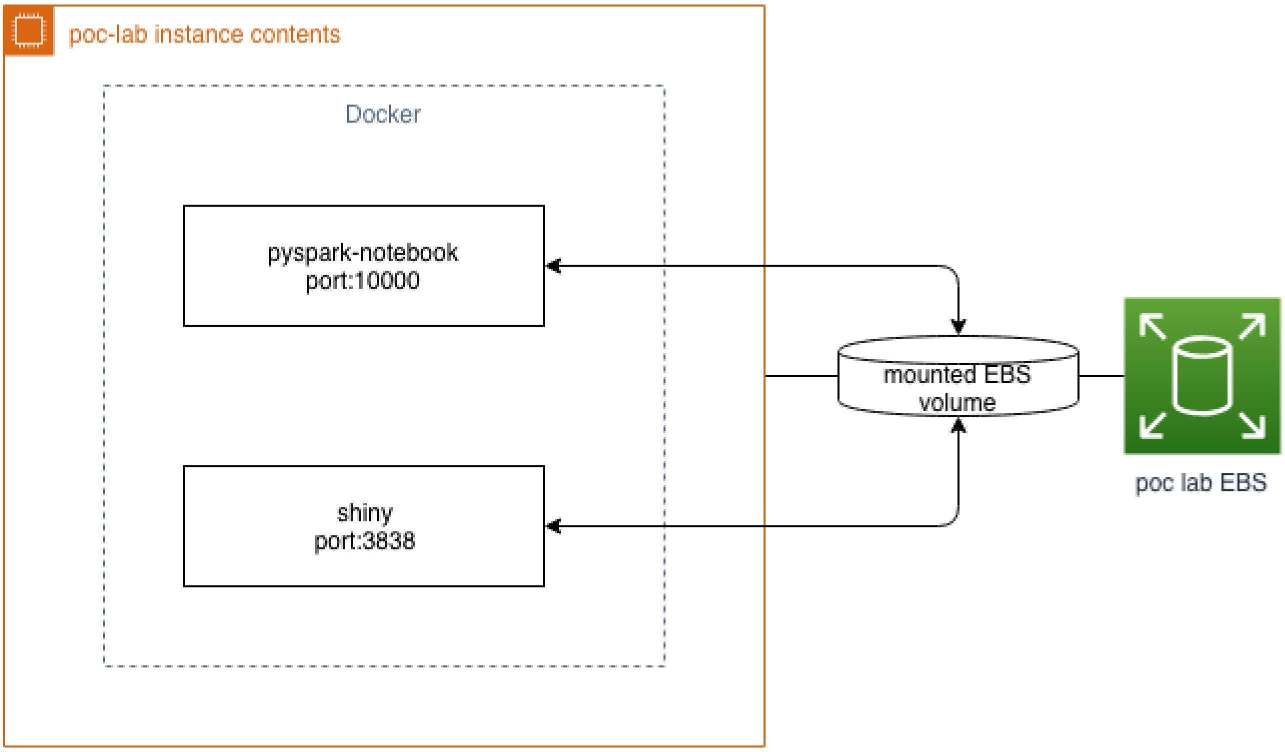
The following section introduces the proposed solution environment for the PoCs. Since the project is in a proof of concept phase, the amount of data is low and security requirements are not stringent, the technical team proposes to deploy the application on a single instance. The diagram below illustrates the environment:



Destination	Target
10.0.0.0/24	local
0.0.0.0/0	internet-gw-id

The environment is deployed in a **Virtual Private Cloud** (VPC) which acts as a private environment for the user. Within the VPC one **subnetwork** is found, the public subnet. This subnet hosts one EC2 instance (server) which has a single EBS volume for storage. The instance has a public IP address and thus is publicly reachable by the user. The instance has a **Security Group** attached, which defines what type of traffic can reach the instance (a detailed description is provided below). The VPC has an attached **Internet Gateway** which allows traffic to the instances and vice-versa. The instance also has access to an S3 bucket, which can be used to persist data and results.

To take into account the scalability requirements, we propose to simulate a compute cluster using docker containers. The application can then be developed using the cluster frameworks, making the code portable to larger compute clusters when the data volume would increase. The proposed docker environment is depicted in the figure below:



The docker environment consists of two docker containers. The `pyspark-notebook` container is built on the official `jupyter/pyspark-notebook` image, which deploys a Jupyter Notebook with a Python kernel and the Anaconda stack, Hadoop and Spark. The container can run Spark in local mode, which simulates a compute cluster. This container will be used for the training phase of the pwc-eac-pilots.

The `shiny` container is built on the `rocker/shiny` image, which deploys Shiny Server with an R installation. This container will be used for the deployment phase of the project.

Containers can utilize any computational and memory resources available so the impact on performance is limited.

Both containers have a mount point on the local file system, allowing them to access files that are stored on the EBS volume that is associated with the instance. This allows the containers to access data that is stored on the file system, store code and write out results to the EBS volume.

Component specifications

The following section aims at providing the specifications of each component in the environment.

pwc-eac-pilots-lab instance

The pwc-eac-pilots-lab instance is an EC2 instance on which the docker environment is deployed. It has the following specifications for the training phase:

Property	Value	Description
Instance type	`m5.xlarge`	AWS Instance type
vCPUs	4	Number of virtual CPU cores
Memory	16	Memory size in GiB

Property	Value	Description
Public IP address	`34.245.7.232`	Only for instances in public network
Private IP address	`10.0.0.7`	
Public DNS	`ec2-34-245-7-232.eu-west-1.compute.amazonaws.com`	Only for instances in public network
Private DNS	`ip-10-0-0-70.eu-west-1.compute.internal`	
Security Group	pwc-eac-pilots-lab Security Group	
Name	`pwc-eac-pilots-lab`	Name tag (can be used to search for instance)

For the deployment phase, the instance can be scaled down, due to less stringent performance requirements:

Property	Value	Description
Instance type	`m5.large`	AWS Instance type
vCPUs	2	Number of virtual CPU cores
Memory	8	Memory size in GiB
Public IP address	TBD	Only for instances in public network
Private IP address	TBD	
Public DNS	TBD	Only for instances in public network
Private DNS	TBD	
Security Group	pwc-eac-pilots-lab Security Group	
Name	`pwc-eac-pilots-lab`	Name tag (can be used to search for instance)

pwc-eac-pilots-lab ebs

Property	Value	Description
Size	200	Size in GiB
Type	`gp2`	SSD general purpose

S3 bucket

You can use the S3 bucket to store persistent data, upload your original data, store results, etc.

The bucket name is `pwc-eac-pilots-dev-eu-west-1`, which you can use with the command line interface (see further).

pwc-eac-pilots-lab Security Group

Ingress

Protocol	Port range	Source addresses/Security group	Description
TCP	22	`0.0.0.0/0`	`ssh` access from outside world to the instance for administration.
TCP	3838	`0.0.0.0/0`	HTTP access to Shiny Server.
TCP	10000	`0.0.0.0/0`	HTTP access to Jupyter Notebook.

Egress

Protocol	Port range	Destination addresses/Security group	Description
ALL	0-65535	`0.0.0.0/0`	Allows traffic to go outside.

PoC 2

The second PoC matches journal articles from various sectors to the course descriptions of universities form PoC 1. The cosine similarity between the course description and the journal articles is calculated.

Requirements

The requirements for the second PoC resemble those of the first PoC, but with less data volume. The stakeholders have expressed that the same environment for the first PoC can be replicated, but using less powerful instances.

Proposed solution

As mentioned before, the same environment from the first pwc-eac-pilots will be replicated.

Component specifications

The following section aims at providing the specifications of each component in the environment.

pwc-eac-pilots-lab instance

The pwc-eac-pilots-lab instance is an EC2 instance on which the docker environment is deployed. It has the following specifications:

Property	Value	Description
Instance type	`m5.large`	AWS Instance type

Property	Value	Description
vCPUs	2	Number of virtual CPU cores
Memory	8	Memory size in GiB
Public IP address	TBD	Only for instances in public network
Private IP address	TBD	
Public DNS	TBD	Only for instances in public network
Private DNS	TBD	
Security Group	pwc-eac-pilots-lab Security Group	
Name	`pwc-eac-pilots-lab`	Name tag (can be used to search for instance)

pwc-eac-pilots-lab ebs

Property	Value	Description
Size	50	Size in GiB
Type	`gp2`	SSD general purpose

S3 bucket

You can use the S3 bucket to store persistent data, upload your original data, store results, etc.

The bucket name is `pwc-eac-pilots-dev-eu-west-1`, which you can use with the command line interface (see further).

pwc-eac-pilots-lab Security Group

Ingress

Protocol	Port range	Source addresses/Security group	Description
TCP	22	`0.0.0.0/0`	`ssh` access from outside world to the instance for administration.
TCP	3838	`0.0.0.0/0`	HTTP access to Shiny Server.
TCP	10000	`0.0.0.0/0`	HTTP access to Jupyter Notebook.

Egress

Protocol	Port range	Destination addresses/Security group	Description
ALL	0-65535	`0.0.0.0/0`	Allows traffic to go outside.

User documentation

Connecting to the instance (Linux/macOS)

Connecting to the instance is done through the Secure Shell (SSH) protocol. Following requirements are necessary:

1. Installed SSH client

Check if your OS has an SSH client installed. An SSH client is required for establishing SSH connections.

2. Public IPv4 address of the instance

The public IPv4 address can be found in the documentation of the instance.

3. Private key with the right permissions

A private key (.pem file) should have been received from the support team via email. Make sure that this private key has its permissions set to 0400 instead of 0777. The path to this key is needed for establishing SSH connections.

Use following command in a command-line shell to set the right permissions:

```
$ chmod 0400 /path/to/my_private_key.pem
```

When all of the above requirements are fulfilled an SSH connection can be made to the instance:

```
$ ssh -i /path/to/my_private_key.pem ec2-user@[public_ip_of_your_instance]
```

A response will be given along the likes of the following:

```
The authenticity of host 'dns_name (ipv4)'
can't be established.
RSA key fingerprint is
1f:51:ae:28:bf:89:e9:d8:1f:25:5d:37:2d:7d:b8:ca:9f:f5:f1:6f.
Are you sure you want to continue connecting (yes/no)?
```

Enter 'yes' in order to complete the connection successfully.

Connecting to the instance (Windows)

The following requirements are necessary:

1. PuTTY

Download and install PuTTY from the PuTTY website.

2. Public IPv4 address of the instance

The public IPv4 address can be found in the documentation of the instance.

3. Private key

Get the full path to your private key (.pem file). PuTTY does not natively support .pem files and requires .ppk. However, PuTTY does have a tool named PuTTYgen that can convert keys to the required format (.ppk).

1. Start PuTTYgen.
2. Under **Type of key to generate**, choose **RSA**
3. Choose the **Load** option.

   By default, PuTTYgen only displays files with the `.ppk` extension. To locate your `.pem` file, select the option to display files of all types.
4. Select your `.pem` file for the private key pair you received.
5. Choose **Save private key** to save the key in the `.ppk` format.

   Save the key without a passphrase when prompted.
6. Specify the same name for the key as for the private key pair you received.

When all requirements are fulfilled a PuTTY session can be started to connect to the instance.

1. Start PuTTY.
2. In the **Category** pane, choose **Session**.
3. In the Host Name field, enter `ec2-user@[instance_public_ipv4]`.
4. Under **Connection type**, select **SSH**.

   Ensure that the **Port** field is set to 22.
5. In the Category pane, expand **Connection**, expand **SSH**, and then choose **Auth**.

   Choose ***Browse*** and select the `.ppk` file that you generated in the previous steps.
6. Choose **Open** to start the PuTTY session.

   When prompted for confirmation to trust the host, choose **Yes**. A new window should pop up that is connected to the instance.

Accessing Jupyter Notebook

The **Jupyter** web application is running in a docker container which has a mounted volume on the instance with path `~\data`. All data stored in this volume is available for **Jupyter**. The web application is accessible in your browser via `http://public_ipv4_adress:10000`. Accessing the user interface requires a security token which can be resolved by entering following command on the instance itself:

```
$ docker exec -ti ec2user_jupyter_1 sh -c "jupyter notebook list"
```

This command will execute a command inside the docker container which runs Jupyter (id=`ec2user_jupyter_1`). The token can be extracted from the response:

```
Currently running servers:
http://0.0.0.0:8888/?
token=52cd36bd251d0e82843fec0ecb99d343a6392b5b4dedcc81 :: /home/jovyan
```

The token in this example is `52cd36bd251d0e82843fec0ecb99d343a6392b5b4dedcc81`.

Accessing R Shiny

R Shiny apps are also running in a docker container and are accessible in the browser via `http://public_ipv4_adress:3838/path/to/app`. Apps need to be stored in the folder of the mounted volume (`~/data`) and are then accessible via that path. For example, if an app is stored in `~/data/myapp/`, the app will be accessible via `http://public_ipv4_adress:3838/myapp/`.

Uploading data/code to the instance

1. Secure Copy Protocol (SCP)

Securely uploading files to the instance is possible through the **Secure Copy Protocol** (**SCP**).

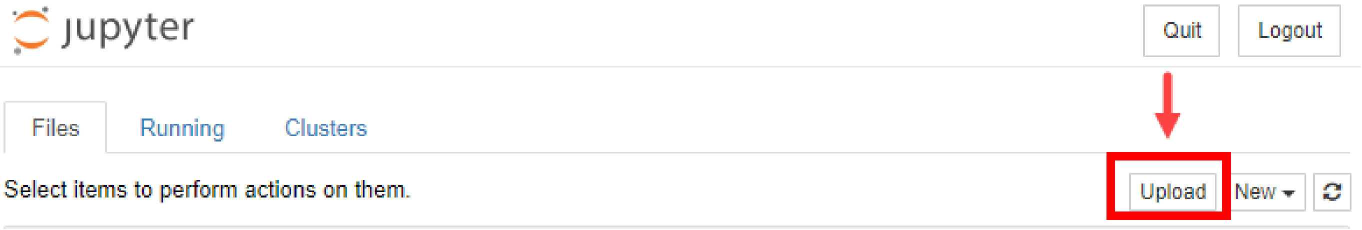
In a terminal use:

```
$ scp -i /path/to/your/private_key.pem FILE ec2-
user@instance_public_ipv4:~/path/on/instance/FILE
```

Make sure the right permissions (0400) are set for the private key.

2. Jupyter

The **Jupyter** web application in the running docker container has access to the mounted volume of the instance (`~/data`). Uploading files is possible by using the Jupyter user interface. All files created and uploaded via Jupyter will be stored on the mounted volume (`~/data`).



Uploading/downloading data to/from the S3 data bucket

The **AWS CLI** is a command line interface to execute API calls from a local machine to AWS that enables to upload data to and download data from the S3 data bucket. Following steps explain this in more detail:

1. Install AWS CLI

A first requirement is that the **AWS CLI** must be installed on your local machine. An installation guide can be found on the AWS website.

2. Configure AWS CLI

Users of the environment receive credentials for using the AWS CLI. The credentials contain an `aws_access_key_id` and an `aws_secret_access_key`. These credentials must be configured as a **profile** in the **AWS CLI**. The following command quickly configures the default profile:

```
$ aws configure
AWS Access Key ID [None]: AKIAIOSFODNN7EXAMPLE
AWS Secret Access Key [None]: wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY
Default region name [None]: eu-west-1
Default output format [None]: json
```

A more extensive guide on how to configure AWS can be found on the AWS website.

3. AWS CLI for S3

If all previous steps have been completed on a local machine the **AWS CLI** can execute API calls to the S3 service.

- Listing all files and folders in the S3 bucket:

  ```
  $  aws s3 ls s3://pwc-eac-pilots-dev-eu-west-1/ --profile default
  ```

- Downloading files from the S3 bucket:

  ```
  $  aws s3 cp s3://pwc-eac-pilots-dev-eu-west-1/FILE FILE --profile
  default
  ```

- Uploading files to the S3 bucket:

  ```
  $  aws s3 FILE s3://pwc-eac-pilots-dev-eu-west-1/FILE  --profile
  default
  ```

Local port forwarding

If accessing the web applications (**Jupyter** or **R Shiny**) is not possible because of your organisation's firewall and/or proxy, using **local port forwarding** is an option.

Following command forwards traffic from a port where the web application is running to a local port through SSH tunneling:

```
$ ssh -i /path/to/your/private_key.pem -N -L
8157:instance_public_ipv4:3838 ec2-user@instance_public_ipv4
```

The web application that normally would be accessible via `http://instance_public_ipv4:3838/` is now accessible through `http://localhost:8157/`. Again make sure that the right permissions are set for the private key.

Docker

Both web applications (**Jupyter** and **R Shiny**) are running in **Docker** containers via `docker-compose`. If for some reason the web applications need to be restarted execute following commands on the instance:

```
$ docker-compose down
$ docker-compose up -d
```

More information on `docker-compose` can be found here.